MASSACHUSETTS INTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-557
September 1989

Boolean Minimization and Algebraic Factorization Procedures for Fully Testable Sequential Machines

Srinivas Devadas and Kurt Keutzer

Abstract

In this paper we continue to investigate the impact of logic synthesis on the testability of sequential circuits that can be modeled as finite state machines. Complete testability of a sequential circuit is ensured by guaranteeing that each invalid state has an unperturbable distinguishing sequence. To accomplish this we present a novel *Boolean minimization procedure of prime implicant generation and contrained covering* based on the Quine-McCluskey algorithm that ensures that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state can be identified. On completion, it guarantees a *prime and irredundant, fully testable Moore or Mealy finite state machine*. Given a two-level circuit with these properties we then use *constrained algebraic factorization* techniques that retain the invariant that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state is detected.

Besides offering a more detailed understanding of the sources of untestability in sequential circuits than previous approaches, this approach offers significant practical advantages as well. It is applicable to a wider range of circuits than optimal synthesis procedures whose utility is often limited by prohibitively high CPU requirements, and its less restrictive synthesis constraints result in lower area overhead than other constrained synthesis approaches. These observations are supported by experimental results.

Microsystems Technology Laboratories | Massachusetts Institute of Technology | Cambridge Massachusetts 02139 | Room 39-321 Telephone (617) 253-0292

Acknowledgements

To be presented at the *International Conference of Computer Aided Design (ICCAD '89)* in November 1989. This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00014-87-K-0825.

Author Information

Devadas: Department of Electrical Engineering and Computer Science, Room 36-848, MIT, Cambridge, MA 02139. (617) 253-0454.
Keutzer: AT & T Bell Laboratories, Murray Hill, NJ 07974. (201) 522-6332

Boolean Minimization and Algebraic Factorization Procedures for Fully Testable Sequential Machines

Srinivas Devadas
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge

Kurt Keutzer
AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract

In this paper, we continue to investigate the impact of logic synthesis on the testability of sequential circuits that can be modeled as finite state machines. Complete testability of a sequential circuit is ensured by guaranteeing that each invalid state has an unperturbable distinguishing sequence. To accomplish this we present a novel *Boolean minimization procedure of prime implicant generation and constrained covering* based on the Quine-McCluskey algorithm that ensures that no single fault can both produce an invalid state *and* corrupt the distinguishing sequence by which that invalid state can be identified. On completion, it guarantees a *prime and irredundant, fully testable Moore or Mealy finite state machine*. Given a two-level circuit with these properties we then use *constrained algebraic factorization* techniques that retain the invariant that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state is detected.

Besides offering a more detailed understanding of the sources of untestability in sequential circuits than previous approaches, this approach offers significant practical advantages as well. It is applicable to a wider range of circuits than optimal synthesis procedures whose utility is often limited by prohibitively high CPU requirements, and its less restrictive synthesis constraints result in lower area overhead than other constrained synthesis approaches. These observations are supported by experimental results.

1 Introduction

Can a sequential circuit be completely tested without adding scan logic? This is perhaps the most open problem in the area of testing. One natural approach to solving this problem is to improve current sequential test generation algorithms. The primary drawback to this approach is that circuit sizes are increasing so quickly that even significant improvements in sequential test generation algorithms cannot keep up. A radically different approach is synthesis for sequential testability. In this approach it is the structure of the circuit itself that is modified to produce fully testable designs.

The idea that logic synthesis and optimization can have a very profound effect on the testability of a synthesized combinational or sequential circuit has been recognized. The relationship between testability and Boolean minimization for two-level and multi-level combinational circuits has been thoroughly investigated [5] [1] [4].

Relationships between sequential logic synthesis and non-scan sequential circuit testability are equally intimate. Scan logic appears to be less necessary for ensuring the testability of datapath portions of circuits because datapath portions have less feedback. As a result, the remaining challenges in synthesizing sequentially testable circuits are to synthesize fully/easily testable control portions and to combine these with datapath portions. Control portions are commonly modeled as finite state machines (FSMs). Synthesis of fully/easily testable FSMs is possible through constrained state assignment and logic optimization [3]. An optimal (sequentially prime and irredundant) synthesis procedure, involving the use of don't care sets in an iterative logic minimization strategy, that produces an irredundant sequential machine with no area/performance overhead was presented in [2].

Our approach represents a middle path between the CPU-intensive optimal synthesis procedure of [2] and the area-penalizing constrained synthesis procedure of [3], combining the advantages of both approaches. Unlike the approach of [2], complex don't care sets do not have to be exploited, nor is repeated logic minimization required. The procedure also does not involve the addition of extra edges or state assignment constraints as in [3]. The FSM is described at the State Transition Graph (STG) level. The optimized logic-level implementation is guaranteed to be fully testable for *all single stuck-at* faults in the combinational logic *without access to the memory elements*.

The approach of this paper is to use synthesis to ensure the complete testability of a sequential circuit implementing a FSM by ensuring that each invalid state has an unperturbable distinguishing sequence. To accomplish this we present a novel *Boolean minimization procedure of prime implicant generation and constrained covering* based on the Quine-McCluskey algorithm that ensures that no single fault can both produce an invalid state *and* corrupt the distinguishing sequence by which that invalid state can be identified. On completion, it guarantees a *prime and irredundant, fully testable Moore or Mealy FSM*. Given a two-level circuit with these properties, we then define *constrained algebraic factorization* techniques that retain the invariant that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state is detected.

Besides offering a more detailed understanding of the sources of untestability in sequential circuits than these previous approaches, this approach offers significant practical advantages as well. It is applicable to a wider range of circuits than optimal synthesis procedures whose utility is often limited by prohibitively high CPU requirements, and its less restrictive synthesis constraints result in less area overhead than other constrained synthesis approaches. These observations are supported by our preliminary experimental results.

Basic definitions and terminology are given in Section 2. Procedures to synthesize fully testable FSMs implemented by two-level and multi-level combinational networks are described in Sections 3 and 4, respectively. The required modifications to the covering step in two-level Boolean minimization are described in Section 5. Preliminary experimental results are given in Section 6.

2 Preliminaries

A cube is written as a bit vector on a set of variables with each bit position representing a distinct variable. The values taken by each bit can be 1, 0 or 2 (or – or don't care), signifying the true form, negated form and non-existence respectively of the variable corresponding to that position. A **minterm** is a cube with only 0 and 1 entries.

A minterm m_1 is said to **dominate** ($m_1 \supseteq m_2$) if for each position that m_2 has a 1, m_1 also has a 1.

A finite state machine (FSM) is represented by its **State Transition Graph** (STG). $G(V, E, W(E))$ where V is the set of vertices corresponding to the set of states S. $||S|| = N_s$ is the cardinality of the set of states of the FSM. An edge joins v_i to v_j if there is any vector of primary input values that causes the FSM to evolve from state v_i to state v_j. $W(E)$ is a set of labels attached to each edge. For the purposes of this paper, we define each label as an ordered 4-tuple $< i, s, s', o >$ where i is a minterm over the primary inputs, s and s' are minterms over the state variables and o is a minterm over the primary outputs. The pair $< s', o >$ corresponds to the output plane of a truth table representation of the FSM. The pair $< i, s >$ corresponds to a minterm in the input plane of a truth-table representation of the FSM; for each edge we will refer to the set of all such pairs as the **input-labels** of that edge.

We denote the primary input combination and present state corresponding to an edge or set of edges is $i \otimes s$, where i and s are cubes over the set of inputs and states respectively. The fanin of a state, q is a set of edges and is denoted $fanin(q)$.

A starting or initial state is assumed to exist for a machine, also called the **reset state**. Given a logic-level finite state machine with N_b latches, 2^{N_b} possible states exist in the machine. A state which can be reached from the reset state via some input vector sequence is called a **valid state** in the STG. The input vector sequence is called the **justification sequence** for that state. A state for which no justification sequence exists is called an **invalid state**. Given a fault F, the State Transition Graph of the machine with the fault is denoted G^F. Two states in a State Transition Graph G are **equivalent** if all possible input sequences when the machine is initially in either of the two states produce the same output response.

A State Transition Graph G_1 is said to be **isomorphic** to another

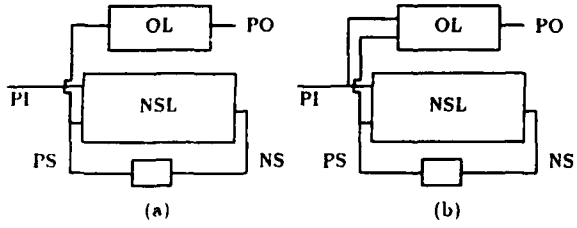

Figure 1: Sequential Machines

State Transition Graph G_2 if and only if they are identical except for a renaming of states.

A primitive gate in a network is **prime** if none of its inputs can be removed without causing the resulting circuit to be functionally different. A gate is **irredundant** if its removal causes the resulting circuit to be functionally different. A gate-level circuit is said to be **prime** if all the gates are prime and **irredundant** if all the gates are irredundant. It can be shown that a gate-level circuit is prime and irredundant if and only if it is 100% testable for all single stuck-at faults [1].

We differentiate between two kinds of redundancies in a sequential circuit. If the effect of the fault cannot be observed at the primary outputs or the next state lines, beginning from any state, with any input vector, the fault is deemed **combinationally redundant**. A **sequentially redundant** fault is a fault that cannot be detected by any input sequence and is not combinationally redundant.

An edge in a State Transition Graph (STG) of a machine is said to be **corrupted** by a fault if either the fanout state or an output label of this edge is changed because of the existence of the fault. A path in a State Transition Graph is said to be corrupted if at least one edge in the path has been corrupted.

Internal single stuck-at faults in a logic network are faults on internal lines (not primary input or primary outputs) that are not equivalent to single or multiple primary output stuck-at faults.

3 Fully Testable Machines with Two-level Logic Implementations

3.1 Introduction

Models for Moore and Mealy machines are shown in Figure 1(a) and (b). Variations of the results below were proven in [2] (*c. f.* Lemma 4.1, Theorems 4.2 and 4.4).

Lemma 3.1 : *Given a reduced Moore or Mealy machine with $N_s \leq 2^n$ states, where n is the number of latches in the machine, all single stuck-at faults on the primary input (PI)/present state (PS) lines and all single and multiple stuck-at faults on primary output (PO)/next state lines (NS) are testable, if the combinational logic of the machine is prime and irredundant.*

Theorem 3.1 : *Given a reduced Moore or Mealy machine with 2^n states, where n is the number of latches in the machine, if the combinational logic of the machine is prime and irredundant and is implemented in two-level form or algebraically factored multi-level form, then the machine is fully testable for all single stuck-at faults in the combinational logic.*

In general, machines will have $N_s \leq 2^n$ states, where n is the number of latches in the machine. Many invalid states may exist that cannot be reached from the reset state of the machine. Invalid states pose a major problem in testability-driven synthesis. A fault may be sequentially redundant, if it *requires* an invalid state to be detected. In order that no fault requires an invalid state to be detected, the invalid state codes have to be used as don't cares in logic minimization. However, if these states are, in fact, used as don't cares, then they may be equivalent to some valid state in G. Thus, we may have a situation where a fault results in a corrupted edge(s) going to an invalid state that is equivalent to the true valid next state. This fault is redundant. More complicated redundancies can be envisioned which involve invalid states that are not equivalent to valid states in G, becoming equivalent to valid states in G^F.

3.2 Fully Testable Moore Machines

The strategy used here modifies the logic minimization process using the invalid states as don't cares, so for each invalid state iv the following conditions are satisfied.

1. *Iv* is not required to detect any fault F in the machine S.
2. *Iv* is distinguishable from any valid state in a specified number (≥ 1) of state transitions or *iv* never appears as a faulty next state, that is equivalent to the true next state.

The goal of the minimization procedure is to satisfy Conditions 1 and 2 *and* produce an area-minimal logic circuit. To this end, we modify the prime implicant generation and covering steps that are basic to two-level Boolean minimization.

Consider the Moore machine of Figure 1(a). We can state the following result.

Lemma 3.2 : *If a reduced Moore machine with N_s states is such that the NSL and OL blocks are prime and irredundant under the invalid state don't care set, and each invalid state has an output distinct from all valid states, then the machine is fully testable.*

Thus, a preliminary minimization strategy is as follows: While minimizing the OL block with the invalid states as don't cares, during covering we select an irredundant set of primes such that all or a maximal number of invalid states have distinct outputs from all the valid states. If we obtain a cover where each invalid state asserts asserts different outputs from all the valid states, then the NSL block can be *unconditionally* minimized with the invalid states specified as don't cares and the resulting machine will be fully testable by Theorem 3.2.

It may not always be possible to perform such a selection. Requiring primality and irredundancy for a cover may conflict with the output requirement. The covering algorithm can instead be made to produce a prime and irredundant cover with a *maximal* set of invalid states asserting distinct outputs from all the valid states. Some invalid states may assert the same outputs as a valid state. For these invalid states, we need to modify the minimization procedure of the NSL block, so as to produce a fully testable machine.

The paradigm followed here is to ensure that the distinguishing sequences, for possible faulty fault-free/state pairs produced due to a fault, are uncorrupted by that fault. These sequences may, of course, be corrupted by other faults. This is accomplished by defining the notion of fault-effect-disjointness (FE-disjointness) between a pair of edges and applying it to two-level combinational networks.

Definition 3.1 : *Given a FSM M, a STG G representing M and a logic-level implementation L of M, a fault f is said to perturb an input-label m of an edge e in G if and only if the fault in L causes the input-label to be removed from e (and moved to another edge).*

Definition 3.2 : *Given a FSM M and a STG G representing M, a logic-level implementation L of M, and two input-labels m_1 and m_2 of two edges e_1 and e_2 in G, the two labels m_1 and m_2 are said to be FE-disjoint over a set of faults $F \in L$ if no fault in F perturbs both m_1 and m_2.*

Definition 3.3 : *A Distance-k-prime-cube (D-k-prime-cube) of a prime cube c is a cube that has exactly the variables of c and a 1 (0) in exactly k positions where c has a 0 (1), in any combination.*

It is only meaningful to talk about a D-k-prime cube relative to a particular prime cube, but whenever the prime cube that is being referred to is unambiguous we will use the term *D-k-prime cube* to abbreviate *D-k-prime cube relative to a prime cube*.

Lemma 3.3 : *Given M, G and a two-level implementation of T of M, and a single internal fault f in T that perturbs an input-label m of an edge e in G, if f is a s-a-0 fault on the output of an AND gate g_i of T then m is contained within the prime cube associated with g_i, and if f is a s-a-1 fault on the input of an AND gate g_j of T then m is contained within a D-1-prime-cube relative to g_j.*

We now state a theorem regarding sufficient conditions for two edge labels to be FE-disjoint over s-a-0 or s-a-1 internal faults in a two-level network.

Theorem 3.2 : *Given M, G and T as above, two input-labels m_1 and m_2 are FE-disjoint over internal s-a-0 (s-a-1) faults in a two-level network, if one of the following conditions is satisfied:*

1. *m_1 and m_2 are not both contained in any prime (not both contained in any D-1-prime-cube) in T.*
2. *m_1 and m_2 are both contained in a prime p_1 (or in a D-1-prime-cube of a prime p_2), and m_1 or m_2 is contained in some other prime p_3 that asserts the same outputs as the prime p_1 (or p_2).*

We are now in a position to define a procedure that produces a fully testable Moore machine.

1. The OL block is minimized with the invalid states used as don't cares, attempting to make sure that a maximal number of invalid states produce different output combinations from all or a maximal number of valid states. If all invalid states produce different outputs from each of the valid states, unconditionally minimize the NSL block and exit. (Two invalid states are allowed to produce the same output).

2. For each invalid state ir_k, find the set of valid states $Q_k = q_{k1}, .. q_{kN_k}$ that assert the same output combination as the invalid state, and such that $ir_k \supset q_{kj}$ or $q_{kj} \supset ir_k$.

3. Perform a two-level Boolean minimization on the logic of the NSL block, meeting the following conditions:

   (a) Use the invalid states as don't cares for all primary input values.

   (b) For each invalid state ir_k, ensure that there exists a PI vector i_{kj} that distinguishes ir_k and $q_{kj} \in Q_k$, $1 \leq j \leq N_k$. That is, i_{kj} produces different next states for ir_k and q_{kj}, such that the next states assert different output combinations, via an appropriate selection of primes. Also, the vector pairs corresponding to $v \in fanin(q_{kj})$ and $i_{kj} \oplus ir_k$ are constrained to be FE-disjoint over (each individual fault in) the s-a-0 (s-a-1) *internal faults in the network corresponding to the cover* if $q_{kj} \supset ir_k$ ($ir_k \supset q_{kj}$), via an appropriate selection of primes that satisfy the conditions of Theorem 3.2.

Theorem 3.3 : *If the procedure above completes successfully, it produces a fully testable Moore machine.*

3.3 Fully Testable Mealy Machines

The procedure is easily extended to the Mealy machine case (Figure 1(b)). The Mealy machine case offers additional flexibility in the choice of distinguishing vectors for any pair of states. We can state the following result.

Lemma 3.4 : *If a reduced Mealy machine with N_s states is such that the NSL and OL blocks are prime and irredundant under the invalid state don't care set and each invalid state is distinguishable in a single state transition from all valid states, then the machine is fully testable.*

The procedure to produce a fully testable Mealy machine is similar to the Moore machine procedure, except that during the minimization of the OL block, we can make choices as to what vectors can be used to distinguish the invalid and valid states, while maintaining primality and irredundancy of the OL block cover. During the minimization of the NSL block, we effectively ensure for state pairs that do not have a distinguishing vector that a two-vector distinguishing sequence for the pair is uncorrupted, if the two states are produced as a faulty/fault-free pair.

4 Fully Testable Machines with Multi-level Logic Implementations

4.1 Introduction

In this section, we extend the results of the previous section to algebraically factored multi-level implementations. Algebraically factored networks are discussed in [4] where it is shown that each single internal fault in an multi-level implementation that was algebraically factored from a prime and irredundant two-level network is equivalent to a multiple internal fault in the two-level network. We therefore begin with perturbation conditions for input-labels under a multiple fault in two-level networks, and then apply these results to algebraically factored networks.

4.2 S-a-0 Faults in a Multi-Level Network

Lemma 4.1 : *Given M, G and T as in Definition 3.1 and a multiple s-a-0 internal fault f in T, if f perturbs an input-label m in G then every prime in which m is contained is affected by the fault. Furthermore, that perturbation results in some next state variable that formerly was 1 to become 0.*

Theorem 4.1 : *Given M, G and T as above, let A be an algebraic factorization of T. Let m_1 and m_2 be two input labels of G and let P_1 be the set of all primes of T that cover m_1 and let P_2 be the set of all primes of T that cover m_2. If both m_1 and m_2 are not contained in any single prime cube in T, and no factor extracted in the factorization of A contains a set of cubes C such that for every prime in p in P_1, some c in C is a subcube of p, and for every prime in q in P_2, some d in C is a subcube of q, then m_1 and m_2 are FE-disjoint over internal s-a-0 faults in A.*

4.3 S-a-1 Faults in a Multi-Level Network

FSMs implemented by multi-level networks are more sensitive to internal s-a-1 faults than to s-a-0 faults. In the case of s-a-0 faults, each input-label m that is a member of the ON-set is covered by some set of primes and for m to be perturbed *all* of those primes must be affected by some s-a-0 fault. If an input-label m is a member of the OFF-set then for each prime p in T there exists a k such that m is contained in a D-k-prime with respect to p, and a multiple s-a-1 fault affecting *any* of the primes in T may perturb m.

Lemma 4.2 : *Given M, G and T as above, and a multiple s-a-1 internal fault f in T, if f perturbs an input-label m in G then m is contained within a D-k-prime relative to an affected prime of T and m is not contained in any other prime of T. Furthermore, that perturbation results in some next state variable that formerly was 0 to become 1.*

Theorem 4.2 : *Given M, G and T as above, let A be an algebraic factorization of T. Let m_1 and m_2 be two input-labels of G. Let f, containing a set of cubes C, be an arbitrary factor in A. If when each literal of each cube of C is expanded in each prime in T in which it appears, there does not exist an expanded prime p in T that covers m_1 and an expanded prime q in T that covers m_2, then m_1 and m_2 are FE-disjoint over internal s-a-1 faults in A.*

These theorems give us constraints sufficient to ensure that no factor extracted from the network results in a network structure in which a single fault perturbs both input labels.

5 The Covering Step

In Section 3, we qualitatively described the requirements to be met during the prime implicant selection or covering step in order to produce fully testable sequential circuits. In this section, we describe the covering algorithm in detail.

One goal of the covering algorithm is to attempt to produce a prime and irredundant cover *under the invalid state don't care set* which has a maximal number of invalid states asserting different outputs from all the valid states. This is achieved via the procedure below, which receives as input the prime implicant table, T, corresponding to the OL block specification.

1. Find all essential prime implicants (EPIs) in T.

2. Pick a (new) invalid state, ir.

3. Find all EPIs in T that contain the invalid state. Let the set of POs that are asserted by *any* of these EPIs, be PO''. We know that the outputs asserted by the invalid state in any prime and irredundant cover has to be $\supseteq PO''$. Pick a (new) $c \supseteq PO''$ for ir (the set of outputs to be asserted by ir) such that:

   (a) Primes that contain ir and assert outputs in $c - PO''$ exist in T.

   (b) c is different from all or a maximal number of valid states that dominate ir or are dominated by ir.

4. Find all primes in T that contain ir and assert outputs not in c, i.e. $\bar{c}$. *Sequentially* delete the rows corresponding to these primes from T, checking after *each* deletion as to whether the next prime to be deleted has become essential due to the previous deletions. If a prime, that is to be deleted, becomes essential, then it means that we cannot find a prime and irredundant cover, where ir asserts c. Go to Step 3 and select a new c. If all choices for c have been exhausted, go to Step 2.

5. Add the invalid states with the chosen outputs as columns to T. This is to ensure that a selection of primes will be made that results in the invalid states asserting the outputs picked at Step 3. (Don't cares are *not* added as columns to the prime implicant table in standard minimization).

6. Solve the covering problem on the modified T using standard heuristic or exact covering algorithms. In order to ensure full testability, when a prime is selected at any stage in the covering, the number of 1s in the output part of the prime is reduced maximally [1]. Also, we do not add a prime to the selected set, unless at least one required vertex is outside the invalid state DC-set and the output part of the prime is reduced taking into account this DC-set.

The procedure described above was for Moore machines. The corresponding Mealy machine procedure is only different in that at Step 3, we choose a primary input vector that can distinguish all or a maximal number of valid states from ir.

[1] This may mean that the selected cube is not a prime in the strict sense.

Once the OL block has been minimized, the NSL block has to minimized obeying constraints similar to those above for the invalid states that assert the same outputs as some valid state(s) (they have to be distinguishable from the valid state(s)). We also have to ensure that the distinguishing vectors detect disjoint sets of faults. The covering procedure is similar to the procedure described above; however, since the NSL block receives inputs from both the primary inputs as well the present states, we have quite some flexibility in choosing distinguishing vectors and the next states produced by the distinguishing vectors.

1. Find all essential prime implicants (EPIs) in T.
2. Pick a (new) invalid state, ir.
3. Pick a (new) input combination pi.
4. Find all EPIs in T that contain $pi \otimes ir$. Let the set of NS lines that are asserted by *any* of these EPIs, be $NS^{pi \otimes ir}$. We know that the NS lines asserted by the invalid state, for this primary input, in any prime and irredundant cover has to be $\supseteq NS^{pi \otimes ir}$. Pick a (new) $c \supseteq NS^{pi \otimes ir}$ for ir (the set of NS lines asserted by $pi \otimes ir$) so:
   (a) Primes that contain $pi \otimes ir$ and assert outputs in $c - NS^{pi \otimes ir}$ exist in T.
   (b) The output of c is different from the output of the next state of all or a maximal number of valid states, V, (that are not already distinguished and which dominate ir or are dominated by ir) on receiving pi.
5. Find all primes in T that contain $pi \otimes ir$ and assert NS lines not in c, i.e. $\bar{c}$. Also, find all primes which contain both $pi \otimes ir$ and $v \in fanin(V)$ or whose D-1-prime-cubes contain both $pi \otimes ir$ and $v \in fanin(V)$. Sequentially delete the rows corresponding to these primes from T, checking after each deletion as to whether the next prime to be deleted has become essential due to the previous deletions. If a prime, that is to be deleted, becomes essential, then it means that we cannot find a prime and irredundant cover, where $pi \otimes ir$ asserts c. Go to Step 4 and select a new c. If all choices for c have been exhausted, go to Step 3.
6. If, at Step 4(b), not all the valid states have different next states from ir on receiving pi, go to Step 3 and attempt to distinguish the remaining valid states from ir.
7. Add the primary input vectors and invalid states with the chosen outputs as columns to T.
8. Solve the covering problem on the modified T using standard heuristic or exact covering algorithms.

At Step 4(b), we only deal at any given pass with valid states that have the same outputs as ir or those that have not as yet been distinguished from ir.

6 Results

In this section, we present preliminary experimental results using the synthesis algorithms presented in Sections 3 and 4.

A standard unconstrained synthesis procedure was first adopted. After synthesis, tests were generated for the circuit using a sequential test generator. Next, we used the synthesis procedure described. After state minimization and unconstrained state assignment, two-level Boolean minimization with constrained covering was carried out. If each invalid state asserted different outputs from all the valid states, then an unconstrained multi-level logic optimization step was performed. Else, two different options of constrained algebraic factorization and unconstrained algebraic/Boolean optimization were exercised. Note that in the latter case, we cannot guarantee 100% testability. The propagation step in sequential test generation is avoided, since we already know all the uncorrupted distinguishing sequences for each possible faulty/fault-free state pair.

We chose some benchmark examples from the MCNC Logic Synthesis Workshop as test cases. The examples had between 24 and 130 states. Results obtained by running the standard synthesis procedure and the two options in the new procedure are summarized in Table 1 under the columns STANDARD, COVER-A and COVER-B. The number of literals in the combinational logic (lit), fault coverage obtained (fcov) and the CPU time for test generation (tpg time) are indicated in the three cases. All the CPU times are on a VAX 11/8800.

COVER-A results in 100% testable designs with small area overheads, that require less CPU time for test generation than the STANDARD procedure. We cannot guarantee full testability via COVER-B, but it allows for the use of more powerful Boolean operations and hence the area overhead is smaller than via COVER-A. Highly (> 99%) testable realizations are obtained in all cases via COVER-B.

EX	STANDARD			COVER-A			COVER-B		
	lit	fcov	tpg time	lit	fcov	tpg time	lit	fcov	tpg time
dfile	171	97.8	5.6m	201	100	3.1m	191	100	3.9m
plan	532	100	2.3m	532	100	2.1m	532	100	2.1m
scf	794	100	82.2m	794	100	43.6m	794	100	73.1m
fs1	432	95.8	25.8m	451	100	11.3m	440	99.8	13.2m
fs2	552	94.9	34.1m	578	100	19.1m	561	99.6	22.1m

EX	COVER			CONSTRAIN			OPTSYN		
	lit	syn. time	tpg time	lit	syn. time	tpg time	lit	syn. time	tpg time
dfile	194	1.4m	3.9m	237[1]	1.3m	15s	165	8.2m	5.5m
plan	532	2.6m	2.1m	568	2.6m	54s	532	2.6m	2.3m
scf	794	4.1m	73m	852	1.1m	81s	794	4.1m	82m
fs1	451	3.7m	11m	544[1]	2.8m	39s	423	1.3h	26m
fs2	578	5.0m	19m	667[1]	3.9m	51s	-	> 2h[2]	-

[1] Involves the addition of an extra input and output.
[2] The synthesis procedure was terminated after 2 hours.

We next compare this approach with previously proposed synthesis approaches to achieve full testability. The comparisons are presented in Table 2. Under the column COVER, we give the result corresponding to COVER-B, if the resulting design was fully testable. Else, we give the result of COVER-A. The column CONSTRAIN has the results obtained by using the constrained state assignment and logic optimization procedure of [3]. The column OPTSYN has the results using the optimal synthesis procedure of [2]. The number of literals in the combinational logic (lit), the CPU time for synthesis (syn. time) and the CPU time required for test generation (tpg time) are indicated. All the designs via each of the procedures are 100% testable.

From the table it is clear that our new approach represents an attractive alternative to either a CPU-intensive optimal synthesis procedure or an area-penalizing constrained synthesis procedure. From the standpoint of CPU usage for minimization and test pattern generation the CONSTRAIN procedure used the least time, but required modifying the original design. The COVER procedure completed all examples with modest to reasonable CPU requirements. The OPTSYN procedure required the greatest amounts of CPU and was prohibitively expensive on one example. Overall, these results indicate that the COVER procedure improves over the previous procedures from the standpoint of quality of result versus CPU time requirements, and more importantly is able to handle designs that the previous procedures could not (without modification).

7 Acknowledgements

The interesting discussions with Pranav Ashar, Tony Ma, Richard Newton, Robert Brayton, Tim Cheng and Alberto Sangiovanni-Vincentelli on sequential circuit optimization and testability are acknowledged. This work was supported in part by the Defense Advanced Research Projects Agency under contract N00014-87-K-0825.

References

[1] K. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. Multi-level Logic Minimization Using Implicit Don't Cares. In *IEEE Transactions on CAD*, pages 723-740, June 1988.

[2] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. Irredundant Sequential Machines Via Optimal Logic Synthesis. In *IEEE Transactions on CAD*, 1989, to appear.

[3] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines. In *IEEE Transactions on CAD*, October 1989, to appear.

[4] G. D. Hachtel, R. M. Jacoby, K. Keutzer, and C. R. Morrison. On the Relationship Between Area Optimization and Multifault Testabilty of Multilevel Logic. In *Proceedings of the International Workshop on Logic Synthesis*, June 1989.

[5] E. J. McCluskey. Minimization of Boolean Functions. In *Bell Lab. Technical Journal*, pages 1417-1444, Bell Lab., November 1956.